

Création d'autorité de certification (CA)

Dans ce livre, nous allons voir comment créer notre propre autorité de certification (CA) pour nos serveurs locaux afin que nous puissions exécuter des sites HTTPS localement sans problème.

- [Pourquoi HTTPS localement ?](#)
- [Comment ça marche ?](#)
- [Installation de votre certificat racine](#)
- [Création de certificats signés par une autorité de certification pour vos sites de développement](#)
- [Conclusion](#)
- [Configuration nginx](#)

Pourquoi HTTPS localement ?

Pourquoi ne pas simplement utiliser le protocole HTTP standard localement ?

Exécuter HTTP alors que votre site de production est uniquement HTTPS est un risque inutile.

Vous vous battrez toute la journée avec des **avertissements SSL** de contenu mixte .

Il est impératif que votre **environnement de développement** reflète le plus fidèlement possible **la production** . 3



Your connection is not private

Attackers might be trying to steal your information from **www.10000-sans.badssl.com** (for example, passwords, messages, or credit cards).
[Learn more](#)

NET::ERR_CERT_AUTHORITY_INVALID



To get Chrome's highest level of security, [turn on enhanced protection](#)

Advanced

Back to safety

La recherche d'une solution SSL locale en ligne vous amènera souvent à vous perdre dans le terrier du lapin des certificats auto-signés. Le principal problème avec les certificats auto-signés localement est qu'ils doivent également être approuvés par votre navigateur. Vous vous retrouvez avec le même message de navigateur, mais cette fois avec `ERR_CERT_AUTHORITY_INVALID`. Cela se produit parce que le navigateur veut vérifier la validité de ce certificat auprès d'une autorité de certification, et ne le peut pas.

La solution est donc de devenir votre propre autorité de certification !

Comment ça marche ?

Pour demander un certificat SSL à une autorité de certification comme Verisign ou GoDaddy, vous leur envoyez une demande de signature de certificat (CSR) et ils vous fournissent en retour un certificat SSL qu'ils ont signé à l'aide de leur certificat racine et de leur clé privée. Tous les navigateurs disposent d'une copie (ou d'un accès à une copie à partir du système d'exploitation) du certificat racine des différentes autorités de certification, ce qui permet au navigateur de vérifier que votre certificat a été signé par une autorité de certification de confiance.

C'est pourquoi, lorsque vous générez un certificat auto-signé, le navigateur ne lui fait pas confiance.

Il n'a pas été signé par une autorité de certification. Pour contourner ce problème, nous générons notre propre certificat racine et notre propre clé privée. Nous ajoutons ensuite le certificat racine à tous les appareils que nous possédons une seule fois, et tous les certificats auto-signés que nous générons seront alors intrinsèquement fiables.

Devenir une (petite) autorité de certification :

“ Il suffit en réalité de deux commandes. Voyons comment nous pouvons procéder sous macOS et Linux, puis voyons comment cela fonctionne sous le système d'exploitation Windows.

1. Génération de la clé privée et du certificat racine sur macOS Monterey et Linux:

Comme macOS et Linux sont tous deux des systèmes d'exploitation de type Unix, les processus de génération des fichiers requis sont identiques. ☐

La seule différence réelle entre les deux est que sur macOS, vous devrez peut-être installer l'application de ligne de commande [OpenSSL](#) . Pour ce faire, si vous ne l'avez pas déjà, installez [homebrew](#) , qui vous permettra d'installer OpenSSL.

```
/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"  
brew install openssl
```

Ensuite, nous pouvons créer un emplacement pour stocker nos fichiers de certificats locaux.

```
mkdir ~/certs  
cd ~/certs
```

Avec cette configuration, nous sommes prêts à générer la clé privée pour devenir une autorité de certification locale :

```
openssl genrsa -des3 -out myCA.key 2048
```

OpenSSL vous demandera une phrase de passe, que nous vous recommandons de **ne pas ignorer** et de conserver en lieu sûr. La phrase de passe **empêchera toute personne qui obtiendrait votre clé privée de générer son propre certificat racine**. Le résultat devrait ressembler à ceci :

```
Generating RSA private key, 2048 bit long modulus  
.....+++  
.....+++  
e is 65537 (0x10001)  
Enter pass phrase for myCA.key:  
Verifying - Enter pass phrase for myCA.key:
```

Ensuite, nous générons un certificat racine :

```
openssl req -x509 -new -nodes -key myCA.key -sha256 -days 1825 -out myCA.pem
```

Vous serez invité à saisir la phrase secrète de la clé privée que vous venez de choisir et à répondre à une série de questions. Les réponses à ces questions ne sont pas si importantes. Elles apparaissent lorsque vous consultez le certificat, ce que vous ne ferez presque jamais. Je vous suggère de définir le nom commun comme quelque chose que vous reconnaîtrez comme votre certificat racine dans une liste d'autres certificats. **C'est la seule chose qui compte.**

Vous devriez maintenant avoir deux fichiers

Félicitations, vous êtes désormais CA. En quelque sorte. ☐☐

2. Génération de la clé privée et du certificat racine sous Windows :

Sous Windows, il est également possible de configurer votre environnement pour exécuter les `openssl` commandes. Il vous suffit de disposer de quelques outils supplémentaires.

Si vous utilisez [Windows Subsystem for Linux \(WSL\)](#), c'est comme si vous utilisiez Linux et les commandes fonctionneront exactement de la même manière.

Le moyen le plus simple de le faire est d'installer [Git pour Windows](#) , qui est fourni avec OpenSSL et l'utilitaire Git Bash. ☐☐

Une fois que vous avez ouvert une fenêtre **Git Bash**, vous pouvez **exécuter les mêmes commandes que pour macOS ou Linux**, avec **une petite différence**. En raison du fonctionnement de certaines applications de console (en particulier OpenSSL) dans Git Bash, **vous devez préfixer toutes `openssl` les commandes à l'aide de l' `winpty` utilitaire.**

Ainsi, par exemple, la commande suivante indique comment générer la clé privée pour devenir une autorité de certification locale dans Git Bash :

```
winpty openssl genrsa -des3 -out myCA.key 2048
```

Les autres petites différences sont les chemins d'accès aux fichiers dans Git Bash. Lorsque vous ouvrez une instance Git Bash, le répertoire de base dans le terminal est mappé à votre répertoire utilisateur dans Windows, mais avec une structure de répertoire de type Linux. Ainsi, si votre répertoire utilisateur est situé dans `c:\Users\nl` Windows, votre répertoire de base Git Bash sera `c/Users/nl` .

Installation de votre certificat racine

Pour devenir une véritable autorité de certification, vous devez obtenir votre certificat racine sur tous les appareils du monde.

Mais nous n'avons pas besoin de devenir une véritable autorité de certification. Nous devons simplement être une autorité de certification pour les appareils que vous possédez.

Nous devons ajouter le certificat racine à tous les ordinateurs portables, ordinateurs de bureau, tablettes et téléphones qui accèdent à vos sites HTTPS. Cela peut être un peu pénible, mais la bonne nouvelle est que nous n'avons à le faire qu'une seule fois. Notre certificat racine sera valable jusqu'à son expiration.

1. Ajout du certificat racine au trousseau macOS Monterey :

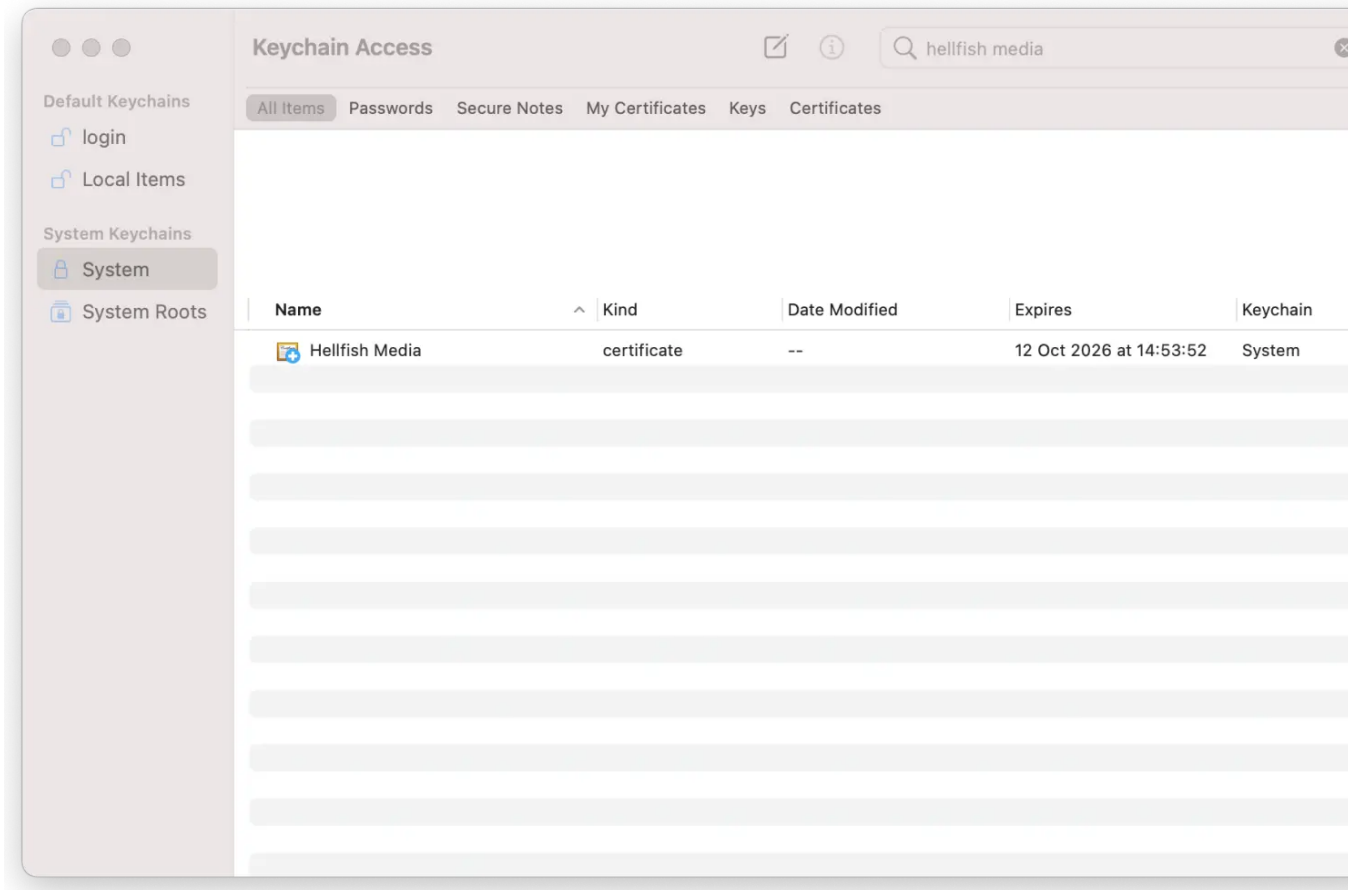
Via le CLI :

```
sudo security add-trusted-cert -d -r trustRoot -k "/Library/Keychains/System.keychain" myCA.pem
```

Via l'application Trousseau macOS :

1. Ouvrez l'application Trousseau macOS
2. Si nécessaire, assurez-vous d'avoir sélectionné le **trousseau système** (les anciennes versions de macOS utilisent par défaut ce trousseau)
3. Allez dans **Fichier > Importer des éléments...**
4. Sélectionnez votre fichier de clé privée (c'est-à-dire `myCA.pem`)


5. Recherchez ce que vous avez répondu comme nom « Nom commun » ci-dessus



6. Double-cliquez sur votre certificat racine dans la liste

7. Développer la section **Confiance**

8. Modifiez la case à cocher « Lors de l'utilisation de ce certificat : » sur **Toujours faire confiance**

 Il est correctement installé, vous verrez les détails du certificat racine.

3. Ajout du certificat racine à Windows 10 :

1. Ouvrez la « Microsoft Management Console » en utilisant la combinaison de touches **Windows + R** , en tapant et en cliquant sur **Ouvrir**
2. Allez dans **Fichier > Ajouter/Supprimer un composant logiciel enfichable**
3. Cliquez sur **Certificats** et **Ajouter**
4. Sélectionnez **Compte d'ordinateur** et cliquez sur **Suivant**
5. Sélectionnez **l'ordinateur local** , puis cliquez sur **Terminer**
6. Cliquez sur **OK** pour revenir à la fenêtre MMC
7. Double-cliquez sur **Certificats (ordinateur local)** pour développer la vue
8. Sélectionnez **Autorités de certification racines de confiance** , cliquez avec le bouton droit sur **Certificats** dans la colonne du milieu sous « Type d'objet » et sélectionnez **Toutes les tâches**, puis **Importer**
9. Cliquez sur **Suivant**, puis **sur Parcourir** . Modifiez la liste déroulante de l'extension de certificat à côté du champ du nom de fichier sur **Tous les fichiers (*.*)** et localisez le fichier, cliquez sur **Ouvrir** , puis **sur Suivant**.
10. Sélectionnez **Placer tous les certificats dans le magasin suivant** . « Magasin des autorités de certification racines de confiance » est la valeur par défaut. Cliquez sur **Suivant** , puis sur **Terminer** pour terminer l'assistant.



Création de certificats signés par une autorité de certification pour vos sites de développement

Nous sommes désormais une autorité de certification sur tous nos appareils et nous pouvons signer des certificats pour tous les nouveaux sites de développement qui nécessitent HTTPS. Tout d'abord, nous créons une clé privée pour le site de développement. Notez que nous nommons la clé privée en utilisant l'URL du nom de domaine du site de développement. Ce n'est pas obligatoire, mais cela facilite la gestion si vous avez plusieurs sites :

```
openssl genrsa -out nl.test.key 2048
```

Ensuite, nous créons un CSR :

```
openssl req -new -key nl.test.key -out nl.test.csr
```

Vous recevrez les mêmes questions que précédemment et, encore une fois, vos réponses n'ont pas d'importance. En fait, elles en ont encore **moins**, car vous ne verrez pas ce certificat dans une liste à côté d'autres.

Enfin, nous allons créer un fichier [de configuration d'extension de certificat X509 V3](#) , qui est utilisé pour définir le [nom alternatif du sujet](#) (SAN) du certificat. Dans notre cas, nous allons créer un fichier de configuration appelé `nl.test.ext` contenant le texte suivant :

```
authorityKeyIdentifier=keyid,issuer
basicConstraints=CA:FALSE
keyUsage = digitalSignature, nonRepudiation, keyEncipherment, dataEncipherment
subjectAltName = @alt_names
```



```
[alt_names]
```

```
DNS.1 = nl.test
```

Nous allons l'exécuter `openssl x509` car la [commande x509](#) nous permet de modifier les paramètres de confiance du certificat. Dans ce cas, nous l'utilisons pour signer le certificat en conjonction avec le fichier de configuration, ce qui nous permet de définir le nom alternatif du sujet.

Nous exécutons maintenant la commande pour créer le certificat : en utilisant notre CSR, la clé privée de l'autorité de certification, le certificat de l'autorité de certification et le fichier de configuration :

```
openssl x509 -req -in nl.test.csr -CA myCA.pem -CAkey myCA.key \  
-CAcreateserial -out nl.test.crt -days 825 -sha256 -extfile nl.test.ext
```

Nous disposons désormais de trois fichiers : `nl.test.key` (la clé privée), `nl.test.csr` (la demande de signature de certificat, ou fichier csr) et `nl.test.crt` (le certificat signé). Nous pouvons configurer les serveurs Web locaux pour utiliser HTTPS avec la clé privée et le certificat signé.

Conclusion

Voilà, vous savez maintenant comment devenir votre propre autorité de certification locale pour signer vos certificats SSL locaux et utiliser HTTPS sur vos sites locaux. Nous espérons que cela éliminera le message redouté « Votre connexion n'est pas privée » sur vos sites Web de développement locaux.

“ crédit

https://deliciousbrains-com.translate.goog/ssl-certificate-authority-for-local-https-development/?_x_tr_sl=auto&_x_tr_tl=fr&_x_tr_hl=fr

Configuration nginx

Générer des paramètres Diffie-Hellman (optionnel mais recommandé) :

Pour améliorer la sécurité, vous pouvez générer un fichier Diffie-Hellman :

```
sudo openssl dhparam -out /etc/nginx/ssl/dhparam.pem 2048
```

Enfin créez un nouveau fichier de configuration pour votre application dans `/etc/nginx/sites-available/` :

```
sudo nano /etc/nginx/sites-available/[NOM]
```

Ajoutez-y la configuration suivante pour utiliser vos certificats SSL (remplacez les chemins de certificats par ceux de vos fichiers) :

```
server {  
    listen 443 ssl;  
    server_name [IP_OU_NOM_DE_DOMAINE];  
  
    ssl_certificate /[EMPLACEMENT_CERTS]/nl.test.crt;  
    ssl_certificate_key /[EMPLACEMENT_CERTS]/nl.test.key;  
  
    ssl_protocols TLSv1.2 TLSv1.3;  
    ssl_prefer_server_ciphers on;  
    ssl_dhparam /etc/nginx/ssl/dhparam.pem; #Sécurité supplémentaire  
  
    #=> Suite de la configuration de votre serveur  
}
```

Enfin activez le site :

```
sudo ln -s /etc/nginx/sites-available/[NOM] /etc/nginx/sites-enabled/
```

Avant de redémarrer Nginx, vérifiez que la configuration est correcte :

```
sudo nginx -t
```

Si tout est correct, redémarrez Nginx :

```
sudo systemctl reload nginx
```